DAMAN CONSULTING

Designing a Data Warehouse

By Michael Haisten

In my white paper *Planning For A Data Warehouse*, I covered the essential issues of the data warehouse planning process.[1] This time I move on to take a detailed look at the topic of warehouse design. In this discussion I focus on design issues often overlooked in implementing several of the warehouse services outlined in my previous article:

- **Acquisition** – bringing data into the warehouse from the operational environment
- **Preparation –** packaging the data for a variety of uses by a diverse set of information consumers
- **Navigation** – assisting consumers in finding and using warehouse resources
- **Access –** supporting direct interaction with data warehouse data by consumers using a variety of tools
- **Delivery** – providing agent-driven collections and subscription information delivery to consumers

Creating the Data Warehouse Data Model

Moral: *A good data warehouse model is a synthesis of diverse non-traditional factors.*
A data warehouse model must be comprehensive, current and dynamic, and provide a complete picture of the physical reality of the warehouse as it evolves. The data warehouse database schema should be generated and maintained directly from the model. The model not just a design aid, but an ongoing communication and management tool.

Managing the dynamic nature of a data warehouse is one of the most daunting challenges. New *feeds* must be supported from new and existing sources. New configurations of data must be packaged with tools to meet continually changing needs. Under-utilized datasets must be retired or consolidated with others. Staying on top of these changes is easier when the model is religiously used to maintain the database structure.

A Hybrid of Concepts, Techniques and Methods

A good data warehouse model is a hybrid representing the diversity of different data *containers*[1] required to acquire, store, package, and deliver sharable data.

To be useful, a warehouse data model must contain physical representations, such as summaries and derived data. The basic entities and attributes that will become core atomic tables are just the starting point for a data warehouse. A purely *logical* model contains none of the substance that defines the primary warehouse deliverables. These deliverables are the many aggregates, packages or collections of data produced to satisfy consumer access needs.

The model must also include *archive data* and *metadata* as well as core data and collections. An archive is a managed, historical extension of the on-line warehouse tables (see "Operations: Implementing a True Archive" below). You must include the definition of the archive tables in the model to record the full context of available data. Metadata is best modeled side-by-side with the data itself and stored co-resident with the warehouse tables.

There is even a more fundamental way in which the data warehouse model is a hybrid. The model is not created using a single methodology. In fact, the better models are a synthesis of differing techniques each contributing a part of the overall result.

Bottom Up Analysis

To design a data warehouse, you must learn the needs of the information consumer. You must decide how the data should be organized. But often designers miss an even more fundamental question, which should come first: What data is *available* for cross-functional sharing and analysis? The question *"What do you want?"* is often answered by the questions *"I don't know, what have you got?"* This is a common circular track in the requirements process. A strict, by-the-book modeler or requirements analyst will loop back with the reply *"Well, I really can't say since what we most need to know is what you know you need."*

Unfortunately, this is the wrong way out of the loop when designing a data warehouse. What data is available often influences what can be done and even what questions are worth asking. What the information consumer *will* need in the future may be more important than what they know they need now. A major part of our job is to help them understand what they don't know they need.

The problem is that we often have no clear picture of what data is locked up in the *data jailhouses* that are our operational systems. Data models are not current, offer insufficient detail, or, more likely, do not exist at all. Trying to interpret the fragmented, denormalized layout of typical legacy systems is a daunting task if done manually.

One technique to consider is the use of a *reverse-engineering tool.* Such a tool reads the legacy system data directly to produce a tentative logical model. Several deliverables are produced which we can use to develop a data warehouse model:

- A consolidated list of unique data elements
- An indication of the fundamental relationships which exist
- A normalized structure which can serve as a starting point for both re-engineering the system and for the design of data warehouse core tables

A candidate list of data warehouse data elements can be produced without any fundamental requirements analysis at all. Start with the consolidated list of data elements from the operational source, then carry out a form of triage. Scan the available data element list and divide them quickly into those that are 1) operational and transitory in nature, 2) of analytic and historical value, and 3) of unknown use or value.

This is easier than it sounds. The first category includes flags and indicators reflecting states of operational processes, control totals, and derived values produced to support transaction processing. Many data elements are replicated from one file to another for performance efficiency.

When in doubt, ask the question, does this element have any lasting meaning after completion of this process? Does it have any historical significance? As an example, a field called *"awaiting shipment"* is clearly one of these transitory indicators. A shipment record contains data of permanent value but *"awaiting shipment"* is worthless after the fact. On the other hand, a field called *"shipment wait duration"* has a potential use after the close of an order. We can use this element to compute a profile of shipment delay times.

A large number of fields or columns in operational system will fall into the first category. The percentage ranges from forty to seventy percent of all fields.

After eliminating the operational and transitory elements, and healthy percentage of the rest should be clearly of analytic or historic value. Elements in the second category are included as candidates for our data warehouse model without question. This includes elements that record the essence of the core business event. Some will become the facts in our data warehouse. Examples include sales order quantity, voucher payment amount, and values like shipment wait duration. Other essential elements of the core business event define the dimensions of analysis such as order date, customer ID, product code, vendor number, etc.

What remains for the third category is often a small proportion of the available data elements (usually between ten and thirty percent).

One principle of warehouse design is to include all category-two elements without insisting on documenting specific current requirements. We suggest taking this principle one step further. We suggest including category three, the indeterminate, as candidates for warehouse inclusion as well. Being a candidate means you must generate solid reasons NOT to acquire these elements for the data warehouse. This turns the traditions requirements process on its head!

Why? Because a common data warehouse failing is to not have what the consumer wants when they come shopping. This is guaranteed to happen if you limit your scope known requirements only. A warehouse must be built to anticipate future needs, and not limited to what is a known need now. And you know what? It is both easier, and less costly in the long run to take it all than to pick and choose!

Top Down Analysis

So how do you anticipate the future? We suggest facilitated brainstorming. Avoid *blank slate* modeling sessions that start from scratch. Avoid a passive requirement process that focuses on revealing known needs. Instead, structure a process that introduces many possibilities and uses strategic business directions to catalyze a conversation about what is known, what may be, and what should be.

This facilitated brainstorming takes the form of several sessions with participation from different constituencies. This process works best when there is a mix of information systems professionals and business information consumers. You start by opening their eyes to the possibilities. You first overview what is known and what is available. Cataloging existing analytic and reporting usage is an effective precursor. If you have completed the bottom up process as discussed above, the candidate data element list is an input to the process. This is used to trigger ideas of new forms of analysis that offer potential benefits to the business.

You continue with a directed process that focuses on the key business and information challenges today that must be overcome. Using published corporate direction statements is often a powerful technique to drive home this message.

An expected deliverable of this process is a weighted or prioritized list of anticipated analytic uses of data. This anticipated usage helps our data warehouse design in two ways:

1. They are used to validate the selection of elements.

2. They provide insight on the data associations that have to be supported in the database.

This second factor directly supports data preparations or the development of specific data packaging to satisfy anticipated needs.

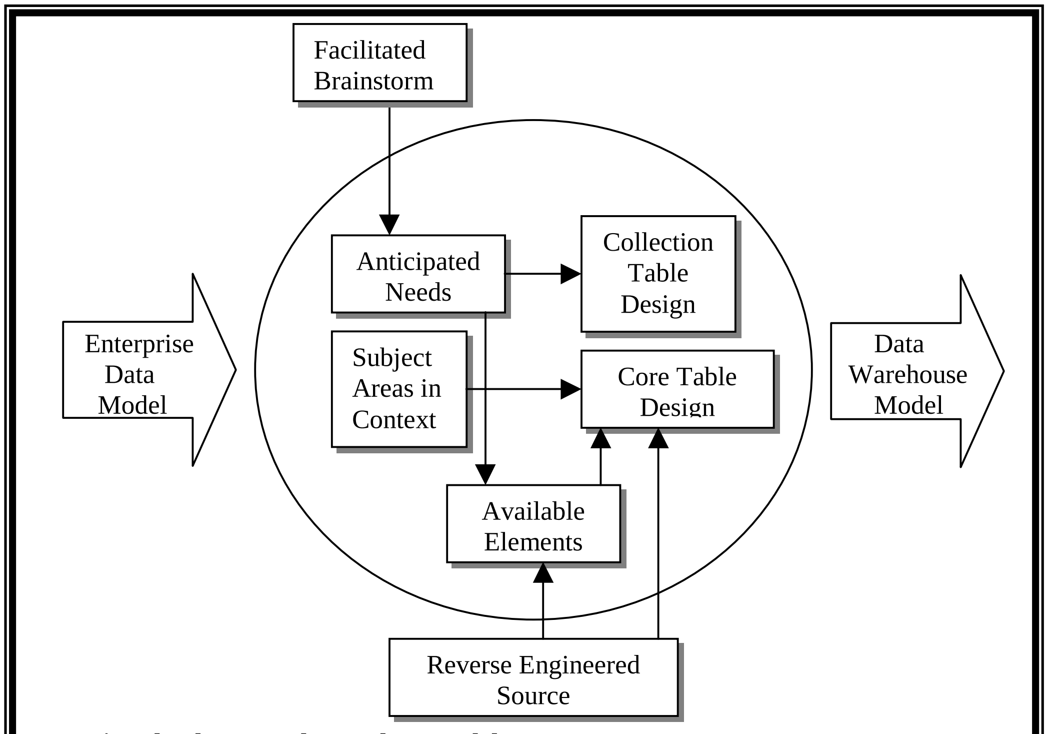
The True Role of an Enterprise Model

There are two flaws with the common perception of the role of an enterprise model in data warehouse design. The first is that an enterprise model is essential. It is not. The second is that the enterprise model is the sole or primary input to the design process. It is only one of many contributing factors. An enterprise model plays a pivotal role in establishing the context and high level structure of a data warehouse data model. But it has very limited impact on the details of the design.

An enterprise model provides a conceptual framework for data integration. It defines the major high-level business entities (which should equate to the warehouse notion of *subject areas*). It depicts how the high level entities interrelate across the full spectrum of the enterprise. Depending on the level of detail, it may suggest critically important business facts that should be included in the warehouse design.

However, an enterprise model is never a sufficient starting point for detailed design. The enterprise entities have insufficient detail to design even the core atomic tables of the warehouse. And the vast bulk of the tables in a warehouse, the ones that take the most effort to design, implement, redesign, and reimplement, are the aggregate and summary collections. This *derived* data would never be represented in an enterprise model, but it is essential to the design of a data warehouse.

The reality is that very few enterprises have a comprehensive and current enterprise model. What exists may not even be sufficient as an integration framework. Worse yet, it may not even serve to identify all the subject areas of the business. You can, and many do, implement an effective data warehouse design without a formal complete enterprise model.



Creating the data warehouse data model

However, in its absence you need some mechanism for defining and bounding your subject areas. You must produce a high-level subject area map to clearly communicate the scope of each subject area and the relationships between subject areas. It is impossible to integrate without having a picture of the whole to which each part relates.

In summary (see Figure 1):

- Use reverse engineering of source systems to define a normalized view of existing data. The resulting logical models are candidates for table design in the warehouse.
- Use the unique data element list from reverse engineering to create a list of candidate elements to be included.
- Conduct facilitated brainstorming sessions to create an ordered list of anticipated data usage. This is used to both validate candidate elements for inclusion and to help define the data associations that will become aggregate and summary collections of data.
- Employ an enterprise model as an integration template revealing the relationship between subject areas. Use it to group tables into subject areas and define macro needs.

Data Acquisition Synchronization Methods

Moral: *Selecting and mapping the source is easy. Synchronizing the target is tough.*

Acquisition services involve getting data out of the enterprise legacy systems and into the base tables of the data warehouse:

- **Extract** – creates a data snapshot from a source system based on identified selection criteria
- **Transport** – sends a snapshot to the location where it will be loaded into the data warehouse
- **Clean** – inspects the data for validity and then optionally corrects the content or marks errors
- **Transform –** implements conversion logic to produce a new value from one or more source fields
- **Populate** – loads warehouse base tables from one or more snapshots

The advent of automated data acquisition tools has fueled the growing interest in data warehousing. Tools such as Carlton Passport, ETI*Extract Tool Suite by Evolutionary Technologies International, and Warehouse Manager by Prism have finally validated the benefits of CASE code generation technology, providing reusable definitions and highly efficient development and maintenance.

Uninformed use of these tools, however, may hide key design decisions implicit in the default techniques used by these products. Specifying mappings and transformation in real world contexts are always more tricky than they appear. Data validation and cleanup is also often a difficult task. The area where these tools are most limited is in the types of *synchronization* choices they provide.

By synchronization, I mean the combination of extract, transport, and population techniques chosen to balance results achieved, performance and resource utilization, while assuring source and target data consistency. This trade-off is affected by the volatility, volume, and value of the data, the means of transport, and the operational batch window for extract processing. The more options you have, and the more flexibility to mix and match techniques, the better off you will be when faced with the inevitable *unique circumstances*.

Extract Methods

The main extract methods used are the following:

Full Copy unloads the entire file or table. A bulk export or copy command is often the most efficient means. Automated tools may use SQL.

Selective Extraction generally uses SQL either directly against the source data or via or a database gateway or transparency layer to create a subset of the source data. For some data source, a proprietary data manipulation language or export specification is more efficient.

Net Changes (selective) is a variant of selective extraction where the criteria is tuned to extract records that have changed or have been added (net changes) since the last execution.

Net Changes (log) determines net changes by reading the log file of the source DBMS or file system instead of accessing the source data directly.

Net Changes (bucket) accumulates net changes in a *bucket* or hold file under the control of the source or producer applications – no external extract process is used with this method.

Triggered Replication is a record or transaction level process that determines new or updated data automatically when the change event occurs.

Differencing is a technique that compares two copies from different time periods to identify new, updated or deleted records. A full copy unload is a precursor to differencing, which uses flat file comparison techniques.

Populate Methods

The main populate methods used are:

Refresh completely reloads the target warehouse table each time. This can be done by either truncating (emptying) the table or dropping and then creating the table. The loading process may either use SQL, a bulk load utility, or proprietary commands.

Logical Update appends rows to the table and additionally supports logical update via effective dates. Use of a single *as-of-date* is the norm, but a *begin-date* and *end-date* pair are occasionally used for reference data.

Physical Update issues an actual update command to modify values in an existing row or to do a physical update should never be used on base tables in the data warehouse. This destroys information by modifying history data.

Refresh operations are by definition set-oriented. Append and logical update operations can be either record-oriented or set-oriented. A set-oriented approach may use either a bulk load utility or proprietary commands. A record-oriented approach is generally not recommended except in conjunction with streaming transport.

Combinations of Synchronization Method

Some of the main combinations synchronization method are discussed below.

Full Copy extract and Refresh populate – this simple mode is over used and should be avoided in most cases. Generally, only small decode tables or reference data should be refreshed and then only if the source contains both active and inactive values. Business facts or event data should always be maintained incrementally. Prior attempts to deliver management information to the desktop, such as information centers, failed due to overuse of image copies of operational data. The lack of continuity and history is one of the flaws a warehouse is intended to fix.

- **Net change (any) extract and Logical Update populate –** This is the most common mode for most warehouse tables. The three variants are noted below:

- **Net change (selective)** is the most universal method, but offers increasing poor performance for larger tables.

- **Net change (bucket)** offers the best performance, but creates an ongoing dependency on the source system. This significantly inhibits the dynamic flexibility of the warehouse to respond to changing needs. Since the extract is not specified in an automated tool, reusability and development efficiencies are not realized.

- **Net Change (log)**, or what is called *changed data capture* by some vendors, provides performance advantages over the selective approach and retains warehouse control lost via the bucket approach. The downside includes specification complexity and limitations on the number of cases where this method can be applied. For example, the file system or DBMS may not offer a log mechanism that captures the details required or the log may not be enabled. Another issue is that the log may not be maintained in a manner that is compatible with data warehouse synchronization, for example, logs may be reset during an operational cycle or after backup and recovery.

Differencing extract and Logical Update populate – Net change methods often involve a file or table scan under control of a file or DBMS manager. Differencing uses extremely fast flat file record comparator techniques and is amazingly efficient when the volatility is low-to-intermediate on moderate-to-large tables. This technique has been used effectively in custom implementations.

Net Change extract and Streaming transport – This technique is useful, and often, essential in situations where very large volumes of data exist on both source and target systems, there is high volatility of data and limited batch processing time on the source system. A continuous end-to-end process is initiated moving extracted data a row or transaction set at a time directly into the target table until all the net changes have been processed for the session.

Replication extract via a DBMS server process – The use of replication servers is generally not recommended for data warehouse data acquisition. The initial round of such products where created to maintain operational integrity and consistency between two remote copies of the data, one a master, the other a slave. They are near real-time transaction processes that perform hard updates and deletes on the target – A *warehouse table is not a replicant of an operational table*. New products are being introduced that support *lazy* (time sequenced, non-real-time) replication of data and allow the preferred logical update mode. These products are more suitable for data warehousing.

Preparation: Packaging the Data for Use

Moral: *It's not just what you have, it's how it's packaged that makes the difference.*
Collect a broad spectrum of core detailed data drawn from multiple sources systems to populate your target subject area(s) in the warehouse. Anticipate the future needs and take the atomic detail that is needed today and of potential use tomorrow. Doing this right is necessary, but does not guarantee success. You will have acquired the right raw materials, but what will make or break your success is how you prepare the data for use.

A Data Warehouse is fundamentally very different from prior attempts at decision support and end user access. One of the most essential differences is the reliance on massively redundant storage schemes. The data is packaged in a variety of ways to separately optimize each form of access and mode of analysis.

Core Warehouse Data, the basic atomic facts, is structured for efficient and rational storage. These tables are the on-line component of the historical archive and the raw materials for producing specialized tables for consumer use. Core data is always stored in a highly normalized fashion.

Standard Collections are aggregates, related sets of information from multiple base tables, and light summaries that support a wide variety of common uses. If core data is the raw materials in a data warehouse, standard collections are the intermediate assemblies. Manufacturers build intermediate assemblies to more efficiently produce a wide array of very specific, often very different, consumer products, from a much small number of standardized building blocks. Likewise, in a data warehouse, standard collections are produced to more efficiently construct a diversity of specialized data structures. You must produce aggregates and summaries to efficiently support high-level analysis. It is prohibitively expensive to produce all summaries directly from the basic core tables.

Standard collections may be normalized, but summarized. They may be denormalized. They may be denormalized by pre-jointing related information or by inclusion of time series arrays. However, regardless of structure, standard collections are generally maintained in relational database tables. This is because they will be used by relational tools to produce consumer collections and by SQL access tools directly fore analysis by those with detailed information needs.

Consumer Collections are the uniquely packaged selective subsets of data optimized for a defined usage. They are constructed with the content, level of detail, and format required for a given purpose. The purpose is defined by a business question posed by a given consumer who intends to use a specific type of analysis with a particular tool in mind. Often the type of analysis and the tool dictate a specialized form or packaging of the data.

The format and structure of consumer collections varies quite widely, including the following:

- **General Denorms** are highly summarized tables containing basic and derived data selected for a specific requirement. These can be employed by general purpose access tools for a wide variety of uses. These can be employed by general purpose access tools for a wide variety of uses. Their disadvantage is that many of these have to be built to satisfy the diversity of uses.

- Historical summaries are implemented as **Time Series** data stored as arrays. Arrays are often far more amenable to use by novice consumers and general-purpose access tools.

- **Star Joins** exploit a single fact table and about three to five composite dimensional tables containing information regarding the rollup hierarchy. This structure was popularized by Metaphor and is not supported by the Red Brick database system and other specialized tools. It is more flexible than hypercubes and more efficient for access, analysis, and drill-down queries than a normalized design.

- **Snowflake** structures are similar to a star join. They also contain a single central fact table, but the dimensions are specified by a set of tables representing the rollup hierarchy and characteristics in a more normalized fashion. This structure is more accessible by a general purpose query tool but does not offer the performance advantages of a star join.

- **Hypercube** is a single dataset containing multiple levels of pre-calculated information. This dataset may be a relational table or a proprietary structure maintained by a particular DSS or EIS tool. Even when stored in a relational table, this structure is exclusively usable by the tool that creates, maintains, and provides access to it. You trade generalized management and accessibility for very high drilldown efficiency. You also lose flexibility since data must be reloaded rather than incrementally updated.

- **Exports** is a general name for flat file datasets created to satisfy the import requirements of a target tool. The tool may be a standalone OLAP server, a client-based DSS system or DBMS, a spreadsheet tool, or virtually any other data-oriented tool.

You need to match the wide diversity of needs with an equal diversity of data packaging. Once size does not fit all. As business needs change, you need to retire obsolete data packages and build new ones. As new tools are introduced, they often come with unique requirements for data and metadata import, storage, and export.

You have three alternatives for handling the broad diversity of needs:

1. Develop an inventory management process to continually evaluate and refine the data products you offer.

2. Invest in new (but quickly evolving) software products that help you manage warehouse aggregates, for example, HP Warehouse Analyzer, Stanford Technologies Metacube.

3. Buy a VERY BIG machine and run all queries off the base tables.

Operations: Implementing a True Archive

Moral: *Maintaining the atoms and building viable history requires a dynamic archive.*
Earlier we stated that one factor which makes a data warehouse unique is the insistence on acquiring atomic level detail. A second factor is the dedication to maintaining enterprise history. Prior decision support and end-user efforts were, at best, only partially successful because they were based only on a current view of incomplete and primarily summarized data.   The clear challenge is to handle the massive volumes that will result from maintaining both atomic and enterprise history data. The solution is to integrate an *archive* as a planned component of your data warehouse infrastructure from day one.

An archive is auxiliary storage managed to maintain a consistent historical perspective by offering automated archival, on-demand retrieval, and transparent view creation. An archive supports access and analysis of information in the extended inventory without regard to what happens to be *on-line* at the time.

Data Containers in an Archive

An archive is organized to manage several different types of data necessary to retain enterprise history:
Atomic History – The row-by-row atomic details that are often archived as the data is acquired from the source systems to build the base fact core event tables.
Context History – Retaining details is not sufficient to reconstruct an accurate picture of what the environment looked like when the data was recorded. Organizations change, rollup hierarchies are in constant flux, coding schemes come and go, and

in general the reference data that defines the information context is highly volatile. Context history is the periodic recording of this volatile data to understand the full information environment (the context) that existed at the time.
With these two broad categories, we have accounted for all the raw data which is captured in our data warehouse. However, to facilitate practical day-to-day use of the archive, we need at least two other primary data containers:
Time Fragments – Time is the dominant dimension in a data warehouse. Many forms of analysis involve either trends or inter-period comparisons. Since an archive is simply an extension of the on-line data warehouse backwards in time, it pays for us to store primary standard collections of lightly summarized data in a time-aware manner. Time fragments are chunks of an extended domain divided into physically separate datasets based on some common periodicity. For instance, we might retain at least seven years of financial data with three years *on-line*. The data in the archive is stored in quarterly time units that can be retrieved separately. In this way, you can reassemble almost any timeframe for analysis selectively.
Strategic Results – Some highly summarized datasets represent business results that are quite significant and are widely reported. Annual operating results, quarterly sales figures by region, and monthly product forecasts are common examples. Any attempts to maintain enterprise history must archive these *end-point artifacts* or final results, which were used to make strategic decisions.

Services Provided by an Archive

An archive should support at least the following types of services:

- **Atomic Archive/Retrieve –** Storing rows of atomic detail in a fashion that allows selective retrieval by defined dimensions or criteria.
- **Context Management –** Periodic storage of related reference data that may optionally be retrieved when either atomic data or time fragments are retrieved.
- **Fragment Management –** Provides a mechanism to define the periodicity of a data domain, store data according to this periodic interval, and retrieve the data for one or more periods.
- **View/Retrieve Table Creation –** Provides a home for data that is retrieved. One method is to create a union view which encompasses the on-line table and one or more identically structured tables of retrieved data. A second method is to dynamically create and populate a single table into which the data is retrieved. An option of this latter method is to load all, or selective contents of the corresponding on-line table into the newly created retrieved data table.
- **Strategic Results Archive/Retrieve –** Stores and reloads strategic results tables.
- **Navigation Services –** an information catalog browsing service that helps that consumer identify what data is available, in what level of detail, at what location, and for what timeframe. To support an archive, you need to identify for what interval of time the data is available on-line vs. in the archive.

Delivery: Defining a Delivery Infrastructure

Moral: *Interactive access is not the whole game. In fact, alone it is not enough.*
The access and analysis tools market is growing by leaps and bounds. For example, Dr. E.F. Codd has coined the term On-Line Analytic Processing (OLAP) and new vendors are attempting to differentiate themselves by introducing pet phrases such as surging, grazing or diving to supplement trending, data mining, and drill-down, which have been around a little longer. What they all have in common is an assumption that consumers want to interact with their data.

Analysis implies interactivity. However, interactive analysis does not require interactive access. Once consumers have grazed, and determined the criteria that provided useful data, they don’t want to graze anymore. They prefer to have the data delivered when they need it.

Fore many information consumers, analysis is not even the goal. Access and presentation are the sole objectives. One of our most pervasive myths is that our primary customer is an information analyst. The vast majority of the direct consumers of a data warehouse will be information clerks and go-betweens who prepare data for other.

Information clerks don’t surf or graze or dive. They select, format, and present. They often follow known paths, repeating identical or similar processes periodically. What will help them most are mechanisms that free up their machine as quickly as possible and that automate any repetitive activity.

Both an information analyst and an information clerk will benefit from a *delivery service* as an alternative to interactive access. With a delivery service, consumers specify what they want, what form they want it in, where they want it delivered, and when they want delivery to take place.

Alternative Delivery Channels

There are two radically different approaches to delivery: Order fulfillment and agent collection. Order fulfillment, sometimes called subscription delivery, allows a consumer to place an order describing the content and format of the desired results and have it scheduled for delivery to a specified location at a given time. The order may be one time or a repetitive, standing order. Agent collection allows consumers to specify data requirements that are acted upon when a defined event occurs or trigger conditions are met. Orders are time-based captive processes. Agents are event-based autonomous processes.

A range of alternative delivery mechanisms exist:

- **Batch SQL:** SQL is exported form any access tool and scheduled for independent execution on the DBMS server. Results are returned by file transfer to a local server or retrieved by the consumer from the remote output folder or directory. The result files are generally raw tab- or comma-delimited text files, but may be post-processed into a specific PC platform standard.
- **Client Timer:** The client access tool may offer a delayed execution capability. At a specified time, the software activates, possibly in background mode, and runs a prepared script or query template. The results are made available on the local client platform in query document form, not just as raw data.
- **Object-Aware Server:** The client tool may submit a query object to be executed at a specified time on a remote batch server. Generally, an administrator facility is available to control execution sequence, priority, error handling and more. The results are returned to the client platform in a query document form. Unlike the client timer option, this mechanism frees up the client machine entirely while offering more uniform control of consumer order processing.
- **Order-Based Agent:** A query template specifying desired content and format is attached to an agent process. The agent is assigned to execute based on a trigger event or condition. An event example is when new sales data is received from a remote site. A condition example is when costs exceed a set amount for a specific department or account. The results are returned as either raw data or a query document.
- **Search-Based Agent:** Information needs are defined in a general, categorical manner. The agent is instructed to inform you when data is found matching your general criteria. It may optionally collect the data itself, or an abstract or sample if one exists. An example is to specify a search for any mention of a competitor's product within several sources. You may further specify that any information on pricing should be retrieved automatically.

Providing both interactive a delivery-oriented access is essential to offer a complete solution. Agents will play a major role in the future of information access. Some believe agent based processing will also play growing role in data analysis.

References

1. Michael Haisten, "Planning for a Data Warehouse," *InfoDB,* Volume 9, Number 1.